DÚ 2 – Parser argumentů programu

- termín: středa **10.4. 12:00** (poledne)

- Cíle:
  - Pokročilé použití šablon
  - Politiky

- Zadání:
  - Vytvořte objektovou knihovnu pro parsování argumentů předaných programu na příkazové řádce. Knihovna musí umožnit uživateli pomocí politik nastavit její detailní chování.

Požadovaná funkčnost

- Podpora přepínačů bez parametrů
  - `tail` **`--help`**
- Podpora přepínačů s typovaným parametrem (řetězce, číslo, ...)
  - `make -f` **`makefile`** `-j` **`20`**
- Podpora typovaných argumentů (řetězce, čísla, ...)
  - `seq` **`10 20 1`**
- Snadná rozšiřitelnost množiny typů, které knihovna umí naparsovat
- Bonus – formátovaný výpis nápovědy programu

Požadované politiky I

- Konvence
  - DOS/Windows
    - `dir /L /A:D`
  - POSIX
    - `ls -s --width=40 -T 8`
  - Java
    - `java -cp . -verbose:jni -ds`
- Co dělat při neznámém přepínači
  - Vyhodit výjimku
  - Ignorovat
  - Považovat za argument
  - Vypsat chybu a skončit

Požadované politiky II

- Co dělat s rozpoznanými přepínači
  - Nechat být
  - Odstranit z `argv` a aktualizovat `argc`
    - Použití u různých knihoven, viz `gtk_init`, `QApplication`, ...
- Co dělat při chybném formátu parametru přepínače nebo argumentu
  - Vyhodit výjimku
  - Vypsat chybu a skončit
- Co dělat s neočekávanými argumenty
  - Ignorovat
  - Vyhodit výjimku
  - Vypsat chybu a skončit
- Parser musí umět pracovat s ANSI znaky (`char`) i s širokými znaky (`wchar_t`)

Příklad použití I

```
std::wstring file_name;
size_t count = 0;
size_t size = 0;
bool verbose = false;
std::vector<std::wstring> argumenty;

parser<POSIX, wchar_t, ...> p;
p.add("-f" nebo "--file" ulož do file_name);
p.add("-c" nebo "--count" ulož do size);
p.add("--size" ulož do size);
p.add(pokud se vyskytne "-v" nebo "--verbose" ulož do verbose true);
p.set_arguments_container(argumenty);

p.parse(argc, argv);
```

- Příklad je pro konvenci POSIX, pro ostatní konvence by mělo být použití obdobné. Kromě vektorů by mělo být možné vkládat argumenty do listů a množin.

Použití II

```
enum prefix_type { KILO, MEGA, GIGA };
prefix_type prefix;

parser<DOS, ...> p;
p.add("/P" ulož do prefix);

p.parse(argc, argv);

Pozn.: hodnota parametru přepínače /P může být jedno z písmen K (KILO), M
(MEGA) nebo G (GIGA). Je tedy nutné zajistit, aby se textová hodnota parametru
přepínače převedla na správnou hodnotu výčtu.
```

- Funkci, která převede text na hodnotu daného typu musí samozřejmě napsat uživatel knihovny. Knihovna by ale měla poskytovat snadno použitelný mechanismus, jak integrovat uživatelské konvertory.

Použití III

```
std::string log_file;

Parser<Java, char_t, odstraň rozpoznané, ...> p;
p.add(“-l” ulož do log_file);

p.parse(argc, argv);
```

```
Po spuštění programu s parametry:

-f soubor1 -l soubor2 soubor3

a spuštění parsování, bude argc a argv stejné, jako kdyby byl program spuštěn s
parametry:

-f soubor1 soubor3
```

Použití IV – Bonus (až 2 body)

```
bool show_help;
enum attribute_type { HIDDEN, DIRECTORY };
attribute_type attribute;

parser<DOS, ...> p;
p.add(pokus se vyskytne "/H" ulož do show_help true, nápověda k přepínači je
     „Show help");
p.add("/A" ulož do attribute, nápověda k přepínači je
     „File attribute. Supported types are:\nH – Hidden\nD – Directory");

p.show_help();
```

```
Výsledek:
/H            Show help
/A:attribute
              File attribute. Supported types are:
              H - Hidden
              D - Directory
```

- Použití pro ostatní konvence by mělo být podobné a výstup by měl odpovídat zvolené konvenci.

Několik hintů I

- Nezapomenout na možnost částečné nebo parciální specializace šablon, takže lze dosáhnout jiného chování pro různé typy, např. pro typ `char` a `wchar_t`.
- Prozkoumat, co je ve skutečnosti `std::string`, `std::stream` atd. a jestli toho nelze využít pro pohodlnou a jednotnou práci s různými typy znaků.
- Všechny kontejnery v STL umožňují zjistit typ prvků, které uchovávají pomocí členského typu `value_type`.
- `std::istringstream` se může hodit (nejen) pro konverzi řetězce na číslo

Několik hintů II

- Kvůli různým typům znaku (`char` a `wchar_t`) budou pravděpodobně i samotné politiky šablonami. Pokud je potřeba specifikovat, že parametrem šablony je opět šablona, je třeba postupovat přibližně takhle:

```
template <template <typename> class  TPolicy, typename TChar> struct trida{
          void do_something()
          {
                    TPolicy<TChar>::do_something();
          }
};

template <typename TChar> struct politika {
          static void do_something() {}
};

trida<politika, char> t;
t.do_something();
```

Kritéria hodnocení

- Čistá a hezká syntaxe i sémantika
- Kvalita navrženého rozhraní knihovny
  - Není třeba si lámat hlavu s tím, jak přesně se má knihovna chovat, jaké všechny formáty argumentů akceptovat a kolik mezer vypisovat. Zadání je schválně „vágní“, takže řešení mohou být značně variabilní.
- Návrh rozhraní jednotlivých politik
  - Politiky musí skutečně implementovat dílčí kroky algoritmů a nikoliv jenom poskytovat konstanty apod., podle kterých se algoritmus bude řídit
- Ošetření výjimečných a chybových stavů
- Bezwarningová zkompilovatelnost (ideálně GCC i MSVC)
- Včastnost odevzdání
- Minimalizace duplicitního kódu (např. kód pro práci s řetězci by měl být napsaný pouze jednou tak, aby pracoval jak s `char`, tak s `wchar_t`

Pokyny pro vypracování a odevzdání

- Součástí řešení musí být soubor `du2_main.cpp`, který ukazuje použití knihovny, zbytek (tj. celou knihovnu) musí tvořit pouze hlavičkové soubory (vzhledem k použití šablon to jinak ani nepůjde). Jeden ze souborů knihovny musí mít jméno `du2_parser.h`, jehož includováním se zpřístupní všechny potřebné třídy a funkce knihovny.
- na začátek každého souboru vložte komentář typu

```
// DU2-ARG.cpp
// Karel Vomacka NPRG051 2010/2011
```

- vaše řešení vložte do Grupíčku – *neposílejte emailem!*
  - správné soubory do správných sloupečků!